Dependabilidade

Conceitos Fundamentais

Prof. Dr. Iaçanã Ianiski Weber

Confiabilidade e Segurança de Software

98G08-4

Índice

Índice

Objetivos da Aula 1 (versão aprofundada)

- Definir dependabilidade com precisão: **serviço**, **estado**, **ameaças** e **atributos**.
- Distinguir **fault**, **error** e **failure** e entender **ativação** e **latência**.
- Introduzir **modelos de falha** (crash/omission/timing/value/Byzantine) e por que isso muda o projeto.
- Conectar baixo nível (HW/FW/SW): concorrência, memória, temporização, ruído físico, configuração.
- Usar métricas mínimas: **MTTF, MTTR, disponibilidade** e confiabilidade sob hipótese exponencial.

Por que dependabilidade importa em Eng. de Computação?

- Sistemas reais são **heterogêneos**: hardware + firmware + RTOS/Linux + rede + serviços.
- Falhas são **inevitáveis** (defeitos de projeto, envelhecimento, ruído, mudanças de contexto operacional).
- O desafio não é "zero falhas", mas **falhas aceitáveis**: frequência e severidade compatíveis com o risco.

Tese central

Dependabilidade é a engenharia de **evitar** (quando possível), **detectar**, **conter** e **recuperar** de condições anormais.

Dois acidentes clássicos (para ancorar conceitos)

- **Ariane 5 Flight 501 (1996)**: falha catastrófica 40s após a decolagem por interação SW+reuso+condição numérica.
- **Mars Climate Orbiter (1999)**: perda da missão associada a erro de unidades e falhas de processo/integração.

Mapeando Ariane 5: fault → error → failure

- **Fault (sistemático)**: suposição/decisão de projeto + reuso de SW do sistema inercial fora do envelope.
- **Error (estado interno)**: exceção/estado inválido no software do sistema de referência inercial.
- **Failure (serviço externo)**: perda de orientação/atitude percebida pelo sistema de guiagem ⇒ comando incorreto ⇒ perda do veículo.

Lição

Uma falha sistêmica raramente é "um bug só": é **fault técnico + fault de processo** (verificação, validação, hipóteses).

Mapeando MCO: fault → error → failure

- **Fault**: inconsistência de unidades (imperial vs métrica) e ausência de barreiras de validação/integração.
- **Error**: estimativas de trajetória incorretas (estado computacional divergente do real).
- **Failure**: inserção orbital em altitude errada / perda da sonda.

Índice

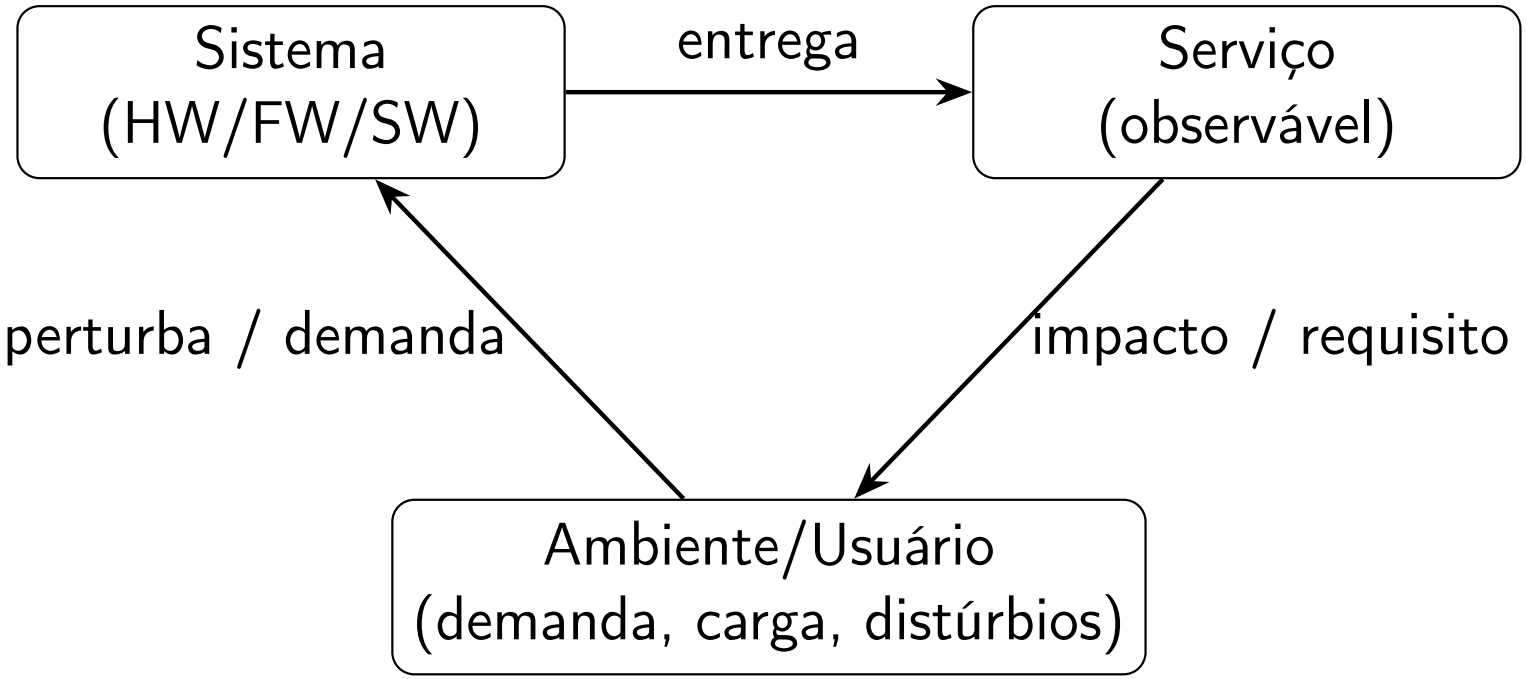
Definição de Dependabilidade (referência clássica)

Dependabilidade

Capacidade de entregar **serviço** que pode ser **justificadamente confiável** (trusted), i.e., evitar falhas mais frequentes/severas do que o aceitável.

- **Serviço**: comportamento percebido na interface com o usuário/outros sistemas.
- **Justificadamente**: exige evidência (processo, análise, testes, monitoramento, certificação, etc.).

Sistema, serviço e estado: o triângulo que organiza tudo



Dependabilidade é sobre manter **serviço correto** sob condições reais, não sobre "nunca errar internamente".

Fault, Error, Failure (com precisão operacional)

Fault Causa potencial (defeito físico, bug, requisito incompleto, operação incorreta). Pode estar **dormente** por muito tempo.

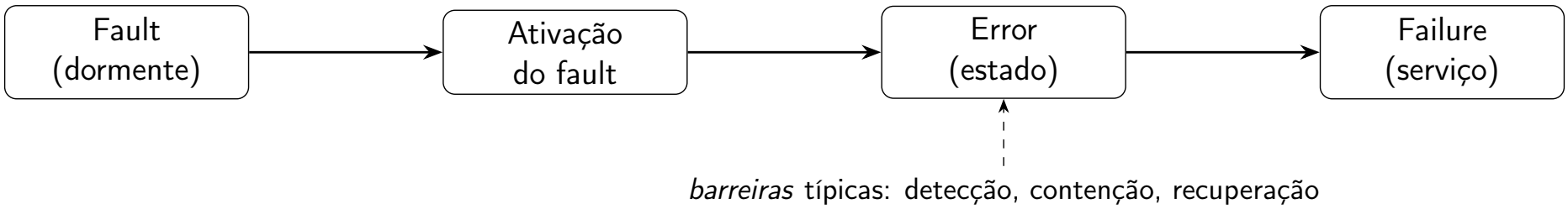
Error Parte do **estado** que é incorreta e pode levar a falha (ex.: bit errado em registrador, estrutura corrompida).

Failure Desvio do **serviço** em relação à especificação (valor errado, atraso, indisponibilidade, comportamento perigoso).

Observação crítica

Nem todo **error** vira **failure**: pode ser sobrescrito, detectado, contido ou mascarado.

Cadeia causal (com ativação e propagação)



Dois pontos onde alunos erram (e custam caro em projeto)

1. **Confundir falha com defeito**: "o sistema falhou" $\neq$ "há um bug" (às vezes é ambiente, operação, integração, requisito).
2. **Assumir determinismo**: em sistemas concorrentes/distribuídos, o fault pode aparecer só sob interleavings raros.

Índice

Atributos: o que exatamente queremos garantir?

Dependabilidade integra atributos (cada um com "pergunta" típica):

- **Disponibilidade**: "o serviço está pronto agora?" (readiness).
- **Confiabilidade (reliability)**: "quanto tempo mantém serviço correto sem falhar?" (continuidade).
- **Safety**: "quando falha, evita consequências catastróficas?"
- **Integridade**: "estado/serviço não é corrompido indevidamente?"
- **Mantenabilidade**: "quão rápido diagnostica, corrige e volta a operar?"
- (Frequentemente junto) **Confidencialidade**: "não vaza informação?"

Disponibilidade $\neq$ Confiabilidade (exemplo intuitivo)

- Um sistema pode ser **pouco confiável** (falha com frequência), mas **muito disponível** (recupera rápido).
- Um sistema pode ser **confiável** (falha raramente), mas **pouco disponível** (reparo demora muito).

Métrica prática

Disponibilidade (regime estacionário):

$$A \approx \frac{\text{MTTF}}{\text{MTTF} + \text{MTTR}}$$

Dependabilidade e Segurança: onde se encontram e onde diferem

- Segurança adiciona adversário e objetivos como **confidencialidade** e **integridade contra ações maliciosas**.
- Há interseção: disponibilidade e integridade importam para ambos.
- Um "fault" pode ser **acidental** (bug) ou **malicioso** (exploit/fault injection).

Índice

Por que "modelo de falha" é indispensável?

Ideia

Projetar tolerância a falhas sem declarar **que tipo de falha** você tolera é como projetar criptografia sem declarar o **modelo de ameaça**.

- Em distribuídos, o que quebra não é "a máquina", é a **assunção** (tempo, rede, consistência).
- Protocolos mudam drasticamente entre **crash** e **Byzantine**.

Classes comuns de falhas (visão de sistemas distribuídos)

Classe	**Descrição resumida**
Crash	processo para (para de responder)
Omission	perde envios/recebimentos (mensagens/ações)
Timing	resposta fora da janela de tempo especificada
Response/Value	responde, mas com valor incorreto
Byzantine	comportamento arbitrário/inconsistente (pior caso)

Semântica de execução: "at most once" é uma promessa forte

O que é RPC?

RPC (Remote Procedure Call) é um mecanismo em que um programa chama uma função/procedimento em outro processo ou máquina, como se fosse uma chamada local.

Em sistemas distribuídos com RPC, após uma falha, pode ter ocorrido:

- chamada não executou,
- executou uma vez,
- executou múltiplas vezes (retries),
- executou parcialmente (efeitos colaterais).

Implicação de engenharia

Para tolerar falhas com *retries*, você precisa de **idempotência**, **deduplicação** e **logs transacionais** (dependendo do caso).

Índice

Faults: classificação útil (para arquitetar mitigação)

- **Físicos** vs **Sistemáticos** (design/requirements/implementation).
- **Internos** vs **Externos** (ambiente, operador, entrada).
- **Permanentes** / **Transientes** / **Intermitentes**.
- **Acidentais** vs **Maliciosos** (quando há adversário).

Baixo nível (HW): soft errors e falhas de memória existem “em produção”

- Erros de DRAM em datacenters foram medidos em larga escala; taxas e distribuição diferem de suposições comuns.
- *Soft errors* (transientes) e *hard errors* (defeitos persistentes) afetam projeto de ECC, scrubbing e observabilidade.

Baixo nível (FW/SW): faults típicos que viram pesadelo

- **Concorrência**: data races, deadlocks, inversão de prioridade.
- **Memória**: use-after-free, buffer overflow, stack corruption.
- **Numéricos**: overflow, underflow, NaNs, saturação.
- **Configuração**: flags, versões, permissões, "feature toggles".

Exemplo: data race (fault) que vira failure só em interleavings raros

```
// Fault: acesso concorrente sem sincronizacao
volatile int counter = 0;

void isr() { counter++; } // interrupcao
int main_loop() { counter++; } // loop principal
```

- **Error**: atualização perdida (read-modify-write não atômico).
- **Failure**: decisão de controle errada (ex.: PWM/atuador fora do esperado).

Faults de temporização: “funciona no laboratório, falha no campo”

- Timing marginal: PVT (process/voltage/temperature) + envelhecimento + carga.
- “Heisenbugs” de tempo: logs mudam a execução; depurar altera o sistema.
- Em RT/controle, timing failure é tão grave quanto value failure.

Índice

Os "4 meios" clássicos para atingir dependabilidade

Meio	Como atua
Fault prevention	evita introduzir faults (processo, arquitetura, restrições)
Fault removal	remove faults existentes (verificação, testes, revisão, análise)
Fault tolerance	entrega serviço mesmo com faults (detecção + recuperação)
Fault forecasting	estima presença/impacto futuro (modelos, dados de campo)

Tolerância a falhas = detectar + conter + recuperar + tratar

- **Error detection**: checks, invariantes, redundância, timeouts.
- **Damage confinement**: isolar região/efeitos (particionamento, MPU/MMU, sandbox).
- **Recovery**: rollback, restart, failover, recomputação.
- **Fault treatment**: remover/neutralizar a causa (desabilitar módulo, substituir, reconfigurar).

Detecção: checks típicos (baixo nível e distribuídos)

- **Checks semânticos**: range checks, sanity checks, invariantes, contratos.
- **Checks estruturais**: CRC/paridade/ECC, checksums em buffers e mensagens.
- **Checks temporais**: timeouts, deadlines, heartbeats.
- **Checks por redundância**: comparação (dual execution), votação (NMR/TMR).

Confinamento: a ideia de "fault containment region"

- Definir domínios onde um fault pode causar dano (processo, container, partição, core).
- Mecanismos: MMU/MPU, privilégios, watchdog por partição, isolamento de barramento.
- Em microserviços: circuit breaker, bulkheads, rate limiting.

Índice

Redundância: três formas fundamentais

- **Redundância de hardware (espacial)**: replicar componentes (ex.: TMR).
- **Redundância temporal**: repetir computação / reexecutar (ex.: lockstep, retry).
- **Redundância de informação**: codificação (ex.: ECC, CRC).

Exemplo HW: ECC de memória como barreira error→failure

- **Fault**: partícula/ruído/defeito altera bits (transiente ou persistente).
- **Error**: palavra lida com bit errado (estado incorreto no datapath).
- **Mitigação**: ECC SECDED corrige 1 bit, detecta 2 bits (dependendo do esquema).
- **Failure evitada**: aplicação não vê dado errado (mas atenção: latência, eventos múltiplos, scrubbing).

Exemplo FW: watchdog + heartbeat (padrão de sistemas embarcados)

- Watchdog reinicia sistema/subsistema se não receber "kick" em janela de tempo.
- Heartbeat monitora vitalidade (thread/processo/sensor).
- Necessita projeto cuidadoso para evitar **reset loop** e para preservar logs de diagnóstico.

Exemplo SW: recovery blocks e N-version programming (diversidade)

- **Recovery block**: executar versão A; se teste de aceitação falhar, rollback e executar B.
- **N-version**: múltiplas implementações em paralelo + votação.

Risco clássico

Falhas de modo comum: se o requisito está errado, todas as versões podem "errar juntas".

Em distribuídos: replicação, consenso e limites fundamentais

- Replicação aumenta disponibilidade, mas exige coordenação (consenso/leader).
- Sob assincronia, há limites teóricos (ex.: impossibilidades e compromissos de liveness/safety).
- Diferença brutal: tolerar **crash** vs **Byzantine**.

Índice

Métricas essenciais (vocabulário de engenharia)

- **MTTF**: tempo médio até falhar (não reparável).
- **MTTR**: tempo médio para reparar/recuperar.
- **MTBF**: tempo médio entre falhas (reparável).
- **ROCOF**: taxa de ocorrência de falhas (failures/time).

Disponibilidade: "novezinhos" e impacto prático

Definição

Disponibilidade é a fração do tempo em que o sistema está operacional/"pronto para serviço".

- 99,9% ("três noves") $\approx$ 8h45m de indisponibilidade/ano.
- 99,99% $\approx$ 52m/ano.
- 99,999% $\approx$ 5m/ano.

Confiabilidade sob hipótese exponencial (modelo simples)

Se a taxa de falha é aproximadamente constante λ (modelo exponencial):

$$R(t) = e^{-\lambda t} \qquad \text{e} \qquad \text{MTTF} = \frac{1}{\lambda}$$

- Útil como primeiro corte (eletrônica/"random failures"), mas **não** modela bem desgaste (*wear-out*).

Exemplo numérico rápido: disponibilidade via MTTF/MTTR

Cenário

Um serviço tem MTTF = 200 horas e MTTR = 20 minutos.

$$A \approx \frac{200}{200 + \frac{20}{60}} = \frac{200}{200.333\ldots} \approx 0.9983$$

- Isso é $\approx$ 99,83% (*downtime* $\sim$ 15h/ano).

RBD (Reliability Block Diagram): série vs paralelo

- **Série**: falha de um bloco derruba o sistema $\Rightarrow R_{sys}(t) = \prod_i R_i(t)$
- **Paralelo**: sistema falha só se todos falharem
  $\Rightarrow R_{sys}(t) = 1 - \prod_i (1 - R_i(t))$

Índice

Exercícios (1/2) — classificação e raciocínio

1. Classifique (fault/error/failure): *timeout no barramento SPI* (dica: pode ser failure; qual fault típico causa isso?).
2. Dê um exemplo de **fault transiente** em SoC e descreva a cadeia fault→error→failure (ou como seria mascarada).
3. Em um firmware com ISR, cite 2 técnicas para **interromper a cadeia** error→failure.

Exercícios (2/2) — métrica e arquitetura

1. Um sistema tem MTTF=1000h e MTTR=2h. Calcule a disponibilidade e estime downtime/ano.
2. Você precisa tolerar **crash** de 1 nó em um cluster de 3 nós. Que classe de protocolo/replicação faz sentido? Quais suposições?
3. Para um sensor crítico, você prefere: (a) TMR ou (b) dual execution + checks? Discuta custo/benefício e falhas de modo comum.

Índice

Resumo da Aula

- Dependabilidade é sobre **serviço correto** e **justificativa** de confiança.
- **fault, error, failure** formam cadeia com ativação e propagação — e há barreiras possíveis.
- Modelo de falha (crash/omission/timing/value/Byzantine) define **o que** significa "tolerar falhas".
- Métricas (MTTF/MTTR/Disponibilidade) permitem argumentar e comparar soluções.

Referências e leituras recomendadas I

- Avizienis, Laprie, Randell, Landwehr. *Basic Concepts and Taxonomy of Dependable and Secure Computing*. IEEE TDSC, 2004.
  https://www.landwehr.org/2004-aviz-laprie-randell.pdf
- Laprie. *Fundamental Concepts of Dependability*. (PDF espelhado em CMU).
  https://course.ece.cmu.edu/~ece749/docs/laprie.pdf
- ETH Zürich. *Reliability of Technical Systems* (slides introdutórios).
  https://www.lsa.ethz.ch/education/vorl/rts/04_rts_introduction.pdf
- Univ. of Edinburgh (SEPP). *Reliability, Availability* (lecture).
  https://opencourse.inf.ed.ac.uk/sites/default/files/https/opencourse.inf.ed.ac.uk/inf2-sepp/2024/lecture23reliabilityavailability.pdf
- Stanford CS244b. *Distributed Systems notes* (failure modes, RPC semantics).
  https://www.scs.stanford.edu/17au-cs244b/notes/intro-print.pdf
- Schneider (Cornell). *Failure models* (capítulo com crash/omission/link).
  https://www.cs.cornell.edu/fbs/publications/MullenderChptr2.pdf
- Ariane 5 Flight 501 Failure Report (Inquiry Board, 1996). (PDF via MIT OCW).
  https://ocw.mit.edu/courses/16-355j-software-engineering-concepts-fall-2005/91f1e550b30b00ad797293f430220f18_ari5fail_ful_rep.pdf
- NASA. *Mars Climate Orbiter Mishap Investigation Board Phase I Report* (1999).
  https://llis.nasa.gov/llis_lib/pdf/1009464main1_0641-mr.pdf
- Schroeder et al. (Google). *DRAM Errors in the Wild* (2009).
  https://research.google.com/pubs/archive/35162.pdf
- Mukherjee et al. *The Soft Error Problem: An Architectural Perspective*. (PDF).
  https://people.csail.mit.edu/emer/media/papers/2005.02.hpca.ser_problem.pdf